Do major travel websites have price discrimination based on user's geolocation? A case study on Kayak.com and Expedia.com

Alex Ipatov[1] and Akhil Gandhasiri[2]

I. Motivation and Need

With the proliferation of Internet technologies and big data, online users are often shown personalized content, whether it is a product recommendation on Amazon.com or a Google search result. This personalization happens thanks to various algorithms that generate results based on users' geolocation, past activity, etc. On one hand, personalization can be advantageous for users – for example, movie recommendations on Netflix. However, there are instances when such personalization can also be at disadvantage to buyers. According to [2], e-commerce websites often customize prices without the user's knowledge, which in some instances leads to some online buyers paying more than others for the same product.

In our project, we were curious to explore whether price discrimination takes place on Kayak.com and Expedia.com, two of the largest travel booking websites in the world. Moreover, we wanted to build a comprehensive automated framework, that would be useful beyond the scope of this project – for example, to find the most favorable geolocation to purchase any e-commerce product.

II. Problem Definition

More concretely, our research question was: do Kayak.com and Expedia.com, two of the largest travel websites in the world, customize prices solely based on user's geolocation, all other factors being held constant?

[1]A.Ipatov is a Masters student in Computer Science, Saint Louis University, N Grand Blvd, St. Louis, MO 63103, USA oleksandr.ipatov@slu.edu

[2]A.Gandhasiri is a Masters student in Electrical and Computer Engineering, Saint Louis University, N Grand Blvd, St. Louis, MO 63103, USA akhil.gandhasiri@slu.edu

III. Research Task

A. Related Work

In the past several years, there have been a few research studies that addressed the e-commerce price discrimination problem.

[2][4] used accounts and cookies of 300 real-world users on multiple popular e-commerce websites to detect the price discrimination. Our study is different because we did not study the effect of user accounts and cookies on price discrimination at all.

[3] focused on detecting price discrimination based on technological differences, personal information, and geographical location. In our study we only focused on studying price discrimination based on geolocation. To achieve the task, we used the NordVPN service, which has servers in about 60 countries, whereas the quoted study deployed several proxy servers at different Planetlab nodes in just 6 cities (two sites in US (east and west coast), Germany, Spain, Korea, and Brazil).

[1] showed that Kayak provides different flights for different prices depending on the location of the device, which can result in savings of $100 for a single ticket. Unlike the author, who did his "research" manually, our goal was to build a comprehensive automated framework for discovering the most favorable (cheapest) geolocation (country) for each <origin_airport, destination_airport, date> tuple.

B. Our Solution

In order to answer the research question, we built an automated framework using Docker, NordVPN command-line interface for Linux, Python and Selenium technologies, that allowed us to

Fig. 1. A location-specific alert popup on Kayak.com

crawl prices of each <origin_airport, destination_airport, date> tuple from every country that had standard NordVPN servers.

For crawling, we included query parameters directly in the HTTP GET request (e.g. https://www.kayak.com/flights/STL-ORD/2021-05-22/?sort=price_a) as opposed to going to https://www.kayak.com and then filling out the search form using various XPaths, which would eventually take us to a similar endpoint as above.

The latter two-step approach would have been much more challenging because it would require proper handling of various alert popups and search form elements, potentially each with a different XPath for a given layout, the latter being unique for region/country. For example, sending a GET request to https://www.kayak.com via a NordVPN server in Spain would redirect us to https://www.kayak.es/?ispredir=true with the website in Spanish and its own layout. A user privacy alert (Fig. 1) would be specific only for the Spanish version of the website and would have a different XPath than a similar alert popup for German version. Thus, we went with a simpler approach by sending the GET request directly to the API endpoint, where we had to deal with a manageable amount of XPaths.

In our study, we only focused on crawling prices for one-way flight tickets.

Fig. 2. A snapshot in time of our Docker state

IV. Implementation Details

NordVPN is one of the largest VPN services in the world and one of the very few that has support for command-line interface on Linux [5]. Users with a subscription can connect up to 6 devices at the same time. We used Docker so that we could crawl data in parallel by spinning up four Docker containers at the same time, each connected to a unique NordVPN server.

We used the NordVPN Docker image [6] that enabled the sharing of its network connection with other containers (in our case, with Selenium containers). As Fig. 2 shows, at one point in time there are 8 active containers – 4 containers based on a NordVPN image and 4 containers based on a Selenium image. Each NordVPN container shares its network connection with its Selenium peer. On Fig.2, the NordVPN containters are connected to VPN servers in Czech Republic, UK, Bulgaria and Portugal respectively.

To achieve persistence of crawled data, we used the Docker Volumes technology, so that the data could persist on disk after the corresponding Selenium container was destroyed.

For this project, we maintained a public repository on GitHub [7]. Although the source code is freely available, we would like to provide a brief description of each individual file's functionality.

A. Dockerfile

The Dockerfile has all the required commands to create our Selenium image. It uses selenium/standalone-chrome as its base image, after which it installs python dependencies required to crawl and parse data. Further, it copies files from the current directory that will be needed to successfully accomplish the crawling task. The Selenium image is built with the

```
docker build -t selenium .
```

command.

B. run_selenium.sh

This is the entrypoint to the Selenium container. Whenever a container is run, this bash script is called. It parses arguments from the "docker run" command and calls an appropriate python script (e.g. kayak_flights.py or expedia_flights.py), passing some of the arguments to it. Moreover, it logs to a txt file the location before and after the execution of the python file with the "curl ipinfo.io" command, as well as python file's stdout. Logging the location via an independent IP address checker is done to provide some level of error checking to ensure that the crawling was indeed performed from a unique geolocation.

C. parse_selenium_args.py

This is the script that contains a helper function that validates and parses command line arguments for kayak_flights.py and expedia_flights.py.

D. kayak_flights.py

This a Selenium script for Kayak.com. As mentioned above, it goes directly to the API endpoint, locates the XPath for the alert popup (Fig. 3) close button, clicks on it, waits for page to load, after which it locates the price results list web element, scrapes all the prices using BeautifulSoup library and writes prices to a txt file, one per line. If successful, a screenshot of the page is stored with _0.png suffix. If something went wrong (either could not close the alert popup or locate the results list web element), the screenshot is taken with the _1.png suffix. The latter distinction between suffixes facilitates the data parsing phase after the crawling phase.

E. expedia_flights.py

Adopts a similar approach as kayak_flights.py with a minor difference that Expedia does not have an alert popup.

F. data.txt

This is a simple txt file that contains data points to be crawled, one per line. Each data point is crawled from every country where NordVPN has servers. For example,

```
01,kayak_flight,BCN,CDG,2021-05-21
```

or

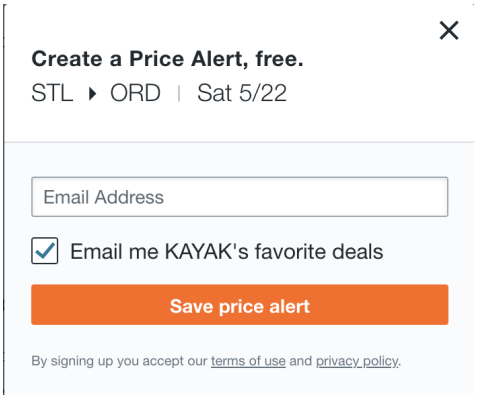

Fig. 3. A location-agnostic alert popup at API endpoint on Kayak.com

```
14,expedia_flight,SFO,JFK,5-21-2021
```

Since Kayak and Expedia have different API design, the flight date is stored in a different format for each. The first number is a unique ID for the flight to facilitate data analysis phase.

G. run.sh

The entire flow is managed with the help of bash scripts. This is the entrypoint to the entire crawling procedure. It takes one argument , "n", – a number of chunks to split the data to allow some level of parallelization. Since, the NordVPN allows up to 6 concurrent connections, the argument cannot be greater than 5 (in the study we went with n=4 just to be safe). For example, the command that we went with was:

```
bash run.sh 4
```

The script validates the command line argument, after which it splits the data into "n" or "n"+1 chunks, depending on whether the data can be split into "n" equal chunks or not. After that, the helper bash script run_helper.sh is called in parallel, depending on the command line argument value. Each call to run_helper.sh gets its corresponding chunk of data from data.txt file. This is achieved with the following command

```
cat data.txt | xargs -n $n_chunks
-P $n_processes bash run_helper.sh
```

where -n and -P flags are crucial to achieve the parallelization effect.

H. run_helper.sh

Each call to run_helper.sh receives its own chunk of data that it is responsible for. All the hard work is done in the double for-loop, as the pseudocode below:

```
for data_point in data_points:
    for country in countries:
        # ...
```

In a nutshell, the server to which the NordVPN container is connected to is changed with the

```
docker exec $container_name
nordvpn c $code
```

command, where "exec" enables to run a command on an already running Docker container [8]. "code" is just a server code supported by the NordVPN CLI (e.g. uk1870).

After that, a corresponding Selenium container is run, which receives its network connection from its NordVPN peer. Once the crawling is done for the <data_point, country> pair, the Selenium container is automatically destroyed (it is accomplished with –rm flag when running the Selenium container) and it proceeds to the next iteration until eventually all data from the chunk is crawled.

V. Results

We share all the crawled data and Jupyter notebook used in the data analysis phase via Google Drive.[1] We crawled information on 688 <flight_id, country> pairs for Kayak.com and 683 pairs for Expedia.com respectively.

For Kayak, 603 pairs were crawled successfully, whereas 85 resulted in errors. The primary two errors were: a) Kayak.com identified us as a bot (Fig. 4) and b) We failed to correctly locate the close button for the alert popup (Fig. 5).

For Expedia, the success rate was significantly lower due to Expedia's superior bot detection measures (see Fig. 6). We managed to crawl prices for 413 <flight_id, country> pairs, whereas 270 resulted in errors.

Before analyzing data in Python, we used Unix tools to split the crawled data into /good and /bad directories based on .png suffixes, as explained above. After that we proceeded with analyzing data found only in /good directories, respectively for Kayak and Expedia. We used Jupyter Notebook on Google Collaboratory for data analysis. The Google Drive folder contains both .ipynb and .pdf files for convenience to see the steps taken to analyze data. In our analysis, we only focused on the cheapest price for each <flight_id, country> pair.

As Fig. 7 shows, out of 12 flight IDs that were used for Kayak, 6 flight IDs had the same price for every country, whereas 6 other flight IDs had two unique prices per flight ID.

As for Expedia (Fig. 8), out of its respective 12

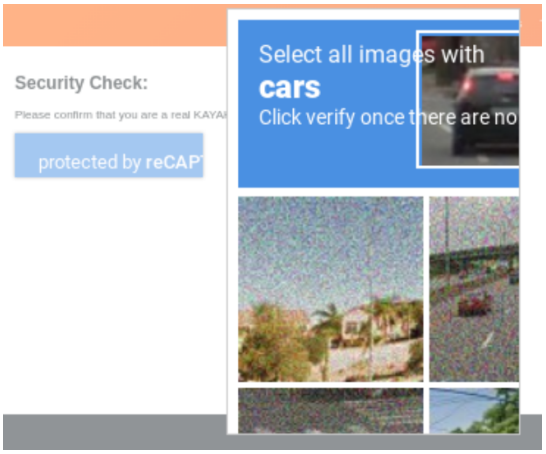

Fig. 4. An example of bot detection on Kayak.com

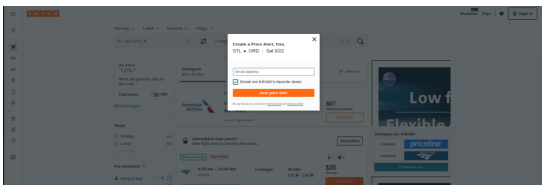

Fig. 5. An example of failing to locate the alert popup close button

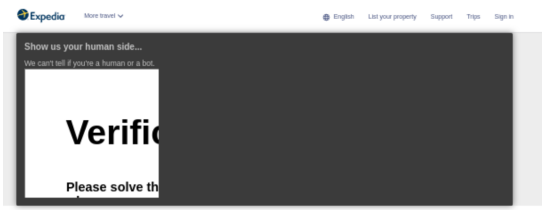

Fig. 6. An example of bot detection on Expedia.com

[1] https://drive.google.com/drive/folders/1-gQg9xM6XAYUfZim_t9kZ_9H-9mlUIyH?usp=sharing

```
Flight_IDs
01       [37, 36]
02          [109]
03         [1681]
04       [55, 47]
05           [25]
06          [389]
07     [383, 465]
08       [58, 69]
09          [389]
10          [499]
11     [240, 235]
12     [200, 201]
Name: Prices, dtype: object
```

Fig. 7. Unique prices for each flight_id, Kayak.com

```
Flight_IDs
13             [48]
14            [108]
15    [3851, 3866]
16             [69]
17             [67]
18            [413]
19            [468]
20             [99]
21            [407]
22            [839]
23            [267]
24            [218]
Name: Prices, dtype: object
```

Fig. 8. Unique prices for each flight_id, Expedia.com

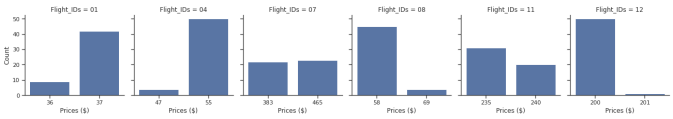

Fig. 9. Frequency of each price category for each Flight_ID, Kayak.com

flight IDs, 11 flight IDs had the same price for every country, whereas 1 flight ID had two prices.

The next logical step was to examine the flight IDs that had more than one price and check frequency of each price category. For Expedia, it turned out that only one country had a different price for flight_ID=15. Thus, it became clear that Expedia had no price discrimination based on geolocation when sending the GET request directly to the API endpoint.

As for Kayak, Fig. 9 shows frequency of each price category for each flight ID. We see that out of 6 flight IDs, 4 flight IDs were largely skewed in favor of one price category. As for the remaining 2 flight IDs, there was a nearly equal frequency between two price categories.

The next logical step was to match each price category with the list of countries for each flight ID (Fig. 10). We could not find a relationship that could potentially explain the difference in prices.

VI. Limitations

The major limitation of our study is sending the GET request directly to the API endpoint during the crawling phase. A much more promising, but also more time consuming, direction would be to send the GET request to https://www.kayak.com and https://www.expedia.com respectively, allowing the backend of those websites to return a customized layout specific for a given region/country. The challenging (and rather boring) part would be dealing with the vast amount of XPaths for different layouts (e.g. to close alert popups, fill out search forms, etc).

On the bright side, those changes can be done directly in the Selenium scripts with no changes to the general architecture that we have built.

VII. Demo

In the end, to show the potential utility of our program, we tested it (with minor changes to the source code on GitHub) on another website, AlgoExpert.io, with a limited amount of NordVPN servers. As Fig. 11 and Fig. 12 show, sending the GET request to https://www.algoexpert.io/purchase via two differ-

Flight_IDs	Prices	
01	36	['cy' 'be' 'is' 'gr' 'al' 'id' 'il' 'my' 'sg']
	37	['cl' 'za' 'hu' 'cz' 'lu' 'dk' 'bg' 'hk' 'es' 'ee' 'no' 'ar' 'pl' 'ba' 'br' 'hr' 'uk' 'it' 'ro' 'tr' 'rs' 'md' 'nz' 'au' 'at' 'ch' 'pt' 'mx' 'se' 'ca' 'vn' 'tw' 'fr' 'th' 'cr' 'de' 'mk' 'kr' 'us' 'ge' 'lv' 'ua']
04	47	['br' 'us' 'ar' 'au']
	55	['se' 'ua' 'uk' 'al' 'tr' 'my' 'ee' 'nz' 'hr' 'bg' 'pl' 'in' 'fr' 'is' 'be' 'at' 'lu' 'cy' 'sk' 'tw' 'za' 'rs' 'dk' 'mx' 'hu' 'cr' 'il' 'nl' 'ie' 'id' 'hk' 'cl' 'vn' 'it' 'jp' 'no' 'kr' 'es' 'ro' 'th' 'lv' 'ca' 'cz' 'ge' 'ba' 'md' 'ch' 'mk' 'fi' 'gr']
07	383	['my' 'id' 'ca' 'hu' 'tr' 'tw' 'nz' 'sk' 'de' 'cr' 'ar' 'cz' 'es' 'al' 'kr' 'ua' 'il' 'no' 'it' 'ba' 'th' 'in']
	465	['lu' 'ee' 'ge' 'pl' 'lv' 'se' 'ro' 'br' 'vn' 'jp' 'md' 'mx' 'be' 'ch' 'rs' 'mk' 'cy' 'za' 'ie' 'fi' 'hk' 'dk' 'au']
08	58	['sk' 'ca' 'sg' 'nl' 'cl' 'mk' 'cz' 'al' 'dk' 'no' 'ba' 'ie' 'tr' 'cr' 'it' 'my' 'fr' 'at' 'vn' 'in' 'ch' 'md' 'lv' 'fi' 'kr' 'za' 'hu' 'tw' 'mx' 'br' 'bg' 'is' 'cy' 'gr' 'pl' 'hr' 'ua' 'rs' 'jp' 'de' 'th' 'nz' 'au' 'il' 'ar']
	69	['ee' 'ro' 'lu' 'hk']
11	235	['au' 'hu' 'my' 'it' 'us' 'cy' 'mx' 'ba' 'ee' 'cl' 'nz' 'dk' 'al' 'bg' 'fi' 'lu' 'pt' 'tw' 'in' 'cr' 'be' 'md' 'nl' 'ro' 'ch' 'es' 'no' 'br' 'sk' 'at' 'pl']
	240	['kr' 'cz' 'sg' 'rs' 'uk' 'is' 'ar' 'ua' 'jp' 'se' 'gr' 'hr' 'za' 'ie' 'il' 'tr' 'ge' 'lv' 'th' 'id']
12	200	['ar' 'mx' 'jp' 'gr' 'au' 'md' 'dk' 'kr' 'de' 'is' 'ge' 'fr' 'hr' 'cy' 'nz' 'ie' 'al' 'ua' 'sg' 'vn' 'za' 'cz' 'rs' 'fi' 'tw' 'nl' 'cr' 'us' 'ee' 'hu' 'il' 'ba' 'hk' 'es' 'mk' 'id' 'no' 'at' 'ch' 'be' 'sk' 'it' 'lv' 'th' 'bg' 'se' 'in' 'my' 'br' 'ro']
	201	['cl']

Fig. 10. A list of countries for each price category, Kayak.com

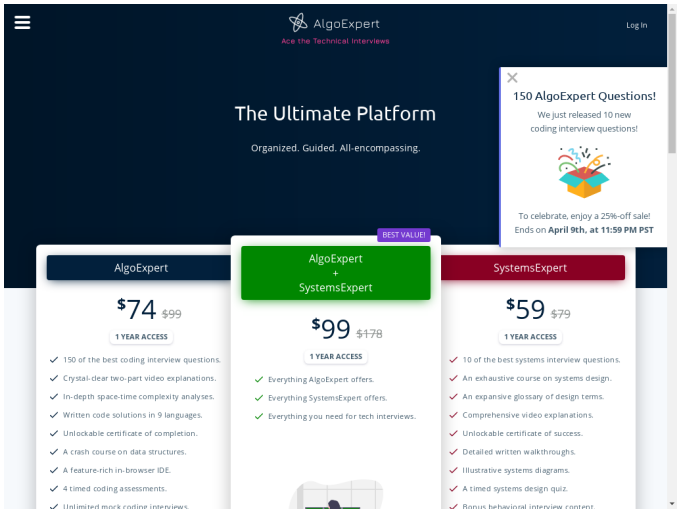

Fig. 11. Annual subscription with the VPN server in the US

ent NordVPN servers can lead to significant savings. Results can be found in the Google Drive folder shared as a footnote on page 4.

VIII. Conclusion

In context of the networks class, while working on this project we gained a better understanding of:

1) VPN technology, as well as its differences from proxy

2) Docker networking

Finally, we hope that the work that we did can be used to find cheapest locations for any e-commerce product by forking and slightly tweaking our source code on GitHub (making it universal rather than specific for Kayak and Expedia), and adding a Selenium script with all the logic and XPath handling for the website to be crawled.

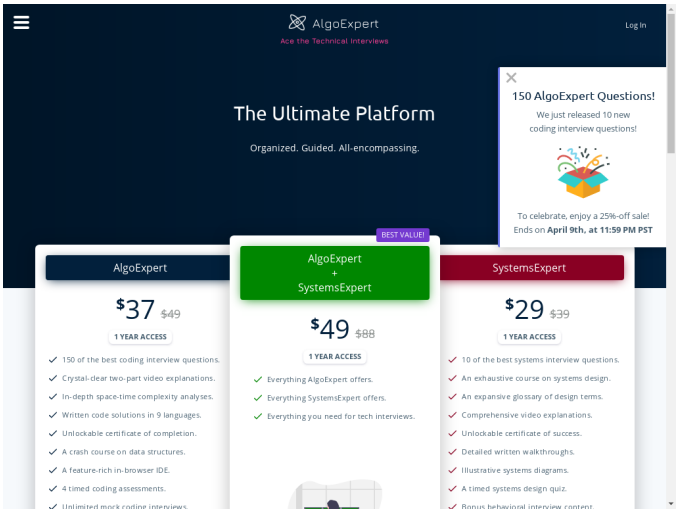

Fig. 12. Annual subscription with the VPN server in Brazil

References

[1] Casanova, J. (2014). How I hacked Kayak and booked a cheaper flight. https://gizmodo.com/how-i-hacked-kayak-and-booked-a-cheaper-flight-1507368539

[2] Hannak et al (2014). Measuring Price Discrimination and Steering on E-commerce Web Sites. Proceedings of the 2014 conference on internet measurement conference.

[3] Mikians et al (2012). Detecting Price and Search Discrimination on the Internet. Proceedings of the 11th ACM workshop on hot topics in networks.

[4] Mikians et al (2013). Crowd-assisted Search for Price Discrimination in E-Commerce: First results. Proceedings of the ninth ACM conference on Emerging networking experiments and technologies.

[5] NordVPN.com (2021). Installing and using NordVPN on Debian, Ubuntu, Raspberry Pi, Elementary OS, and Linux Mint. https://support.nordvpn.com/Connectivity/Linux/1325531132/Installing-and-using-NordVPN-on-Debian-Ubuntu-Raspberry-Pi-Elementary-OS-and-Linux-Mint.htm

[6] https://github.com/bubuntux/nordvpn

[7] https://github.com/satrancci/cheap-ticket-finder

[8] https://linuxcommandlibrary.com/man/docker-exec